Marmara University

Engineering Faculty

Computer Engineering

Cse 2046 Analysis of Algorithms

Project 1 Report

Doç. Dr. ÖMER KORÇAK

Group Members

Elif Nur Kemiksiz	100217006
Muhammet Eren Atala	150119904
Nesrin Şimşek	150119664

Indexes

Purpose

In this project, multiple sorting algorithms will be examined and compared. A java project has been created to make this comparison.

Summary

In this java project, 10 different length lists were created and tested for 5 different k elements. One first index, one last index, and one middle index are selected from the list. One of the remaining two is selected randomly from the first half of the list and the other from the last half of the list. Also Benchmark Library was used as an auxiliary library. Thanks to the benchmark library, the working times for which we will analyze the data have been reached.

Algorithms

Every Iteration has 5 different K values.

Iteration 1, 6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 61, 62, 71 -> First Index of list

Iteration 2, 7, 12, 17, 22, 27, 32, 37, 42, 47, 52, 57, 62, 67, 72 -> Random Index from first half of the list (except first index)

Iteration 3, 8, 13, 18, 23, 28, 33, 38, 43, 48, 53, 58, 63, 68, 73 -> Middle Index

Iteration 4, 9, 14, 19, 24, 29, 34, 39, 44, 49, 54, 59, 64, 69, 74 -> Random Index from last half of the list (except first index)

Iteration 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75 -> Last Index of list

Every Iteration has a different list order.

Iteration 1 – 25 -> Random List

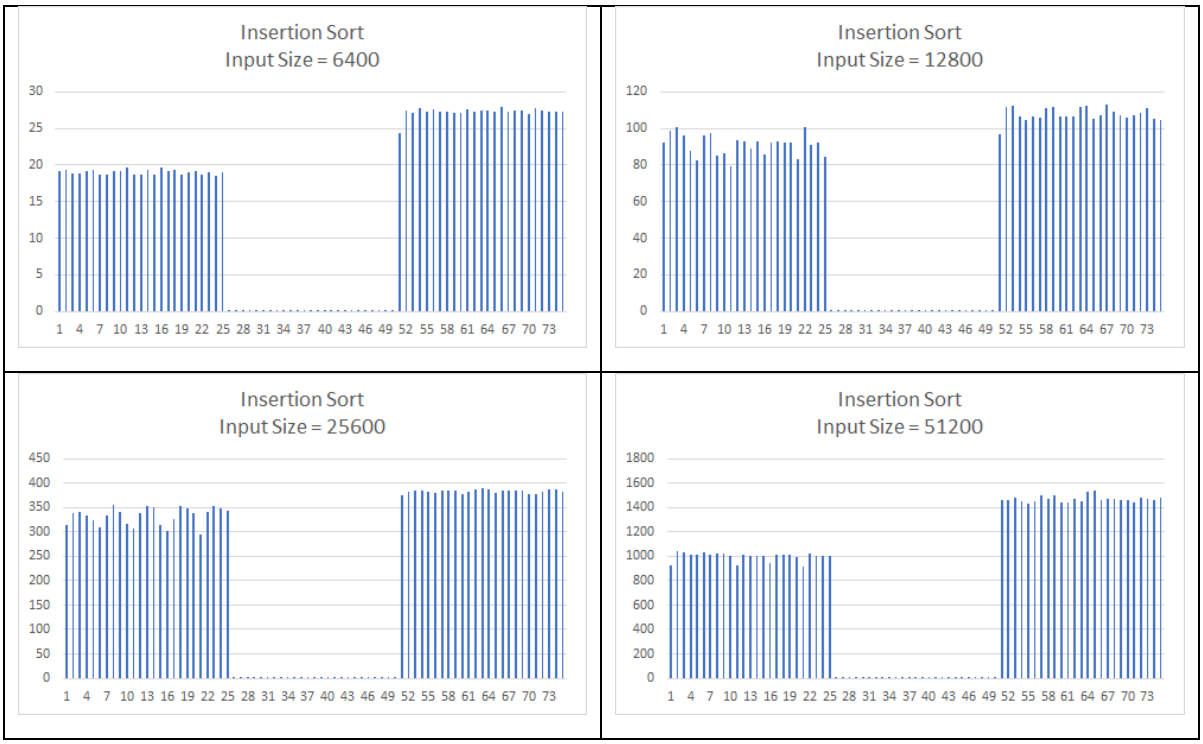
Iteration 26 – 50 -> Ordered Input

Iteration 51 – 75 -> Reverse Input

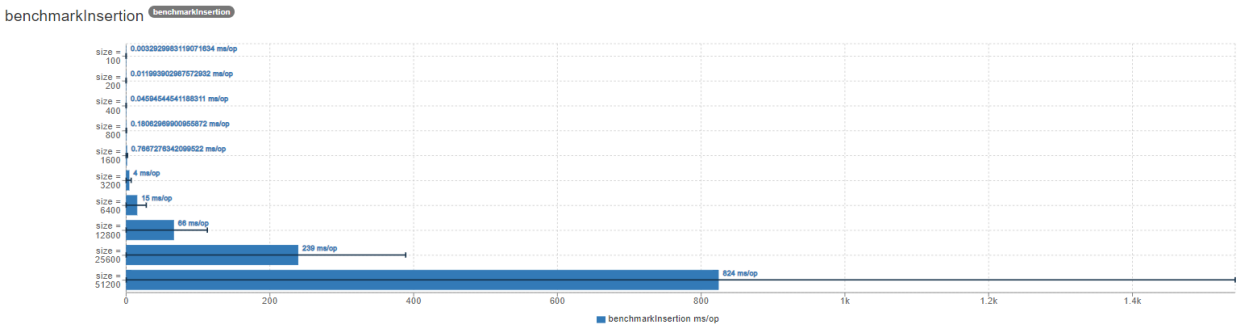
And every combination has a different list size. 100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600, 51200

1. Insertion Sort

The main element that changes the execution time in the insertion sort algorithm is the sorting of the input. The first 25 iterations were run with random input, the middle 25 iterations with ordered input and the last 25 iterations with reversed input. As can be seen in the graphs, there is not much time difference between the random and reversed input, while the execution took a very short time for the ordered input. Thus, it can be deduced that the ordered input is the best case for the insertion sort algorithm. In addition, the K index has no effect on execution time, since sorting is done first and then the K'th smallest element is reached. In addition, every time the input size doubles, the execution time has grown by n^2 for the worst case and average case, and by n for the best case.

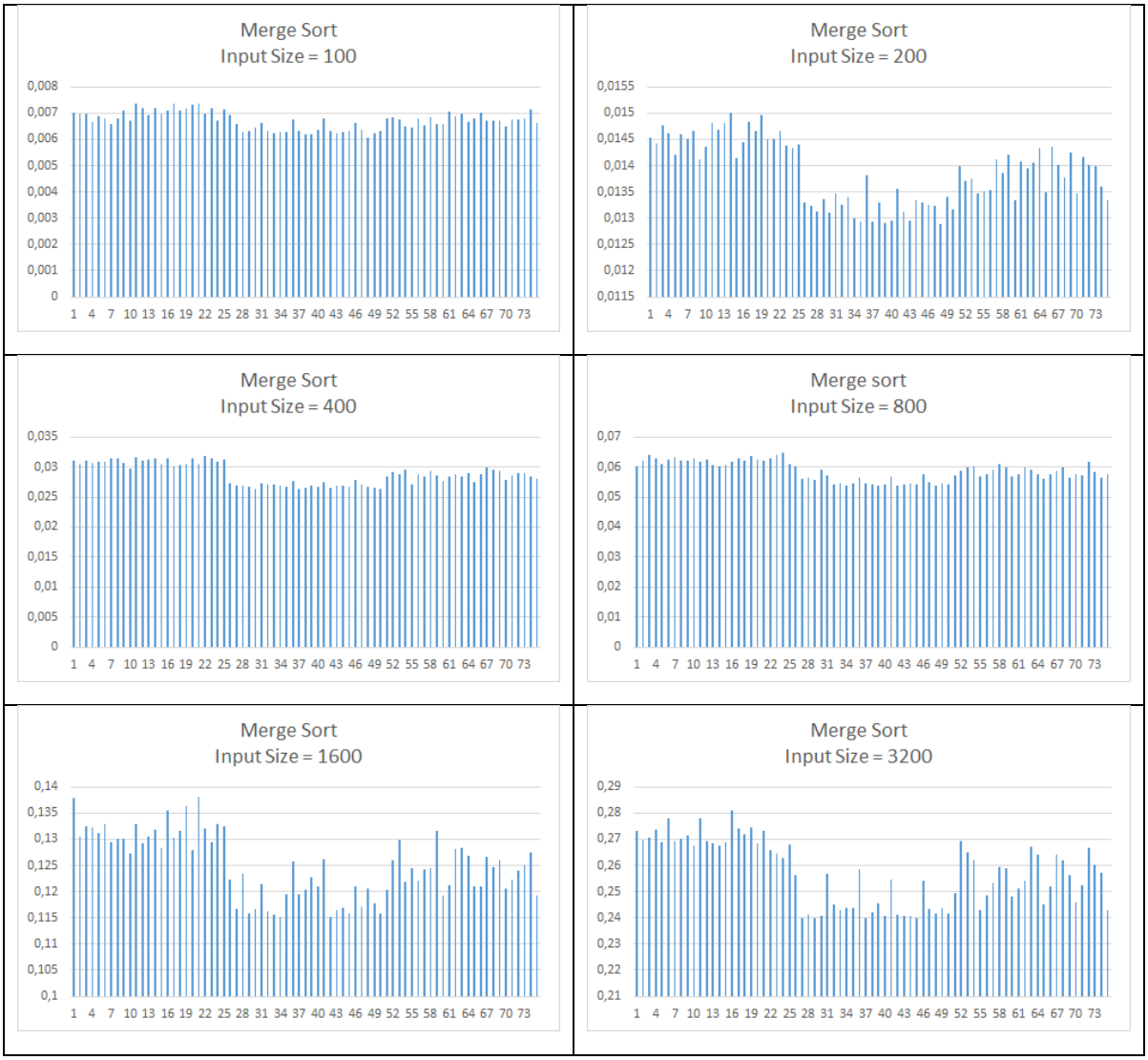


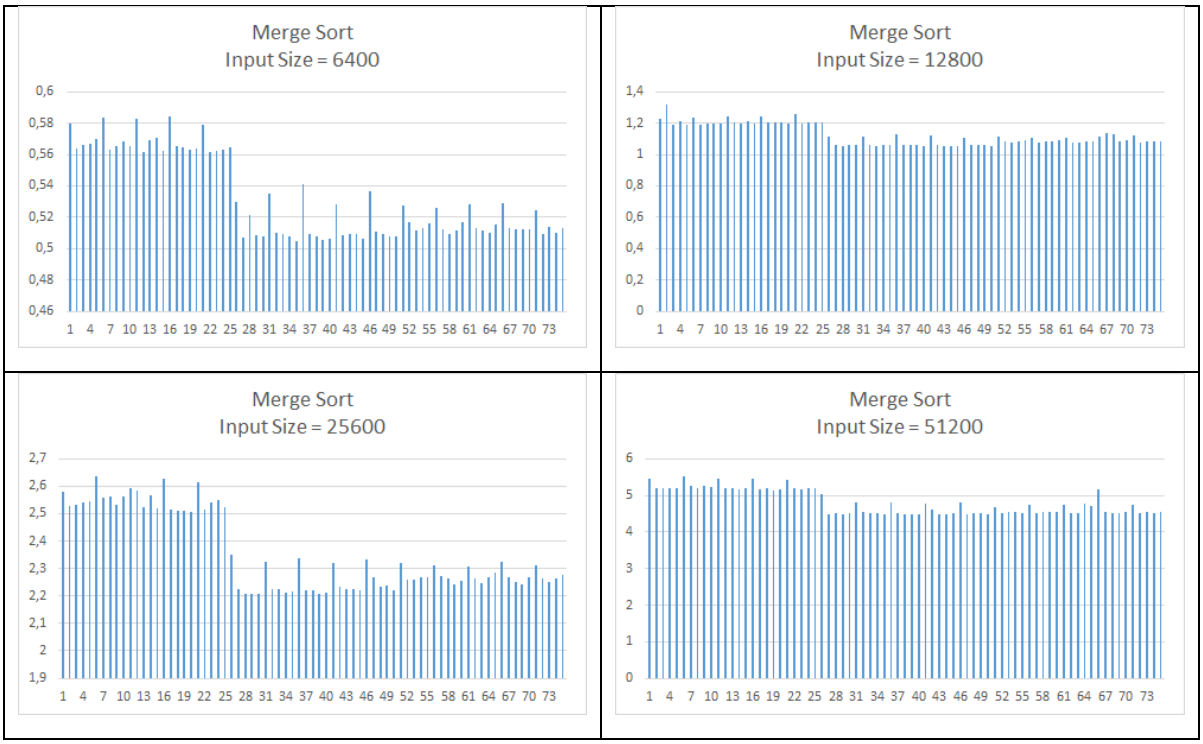
Insertion Sort Average Time Complexity = O(n^2)



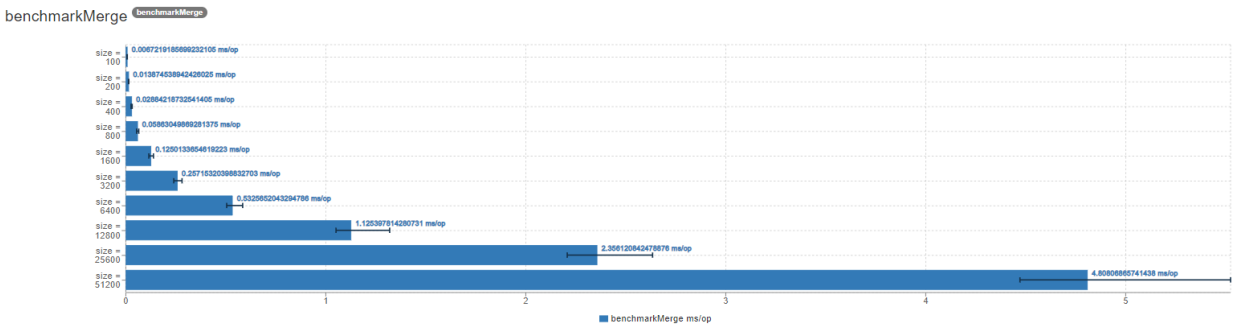
2. Merge Sort

The use of random, ordered and reverse inputs in the merge sort algorithm did not have much effect on execution time. The execution time values for each order gave close results. In addition, the K index has no effect on execution time, since sorting is done first and then the K'th smallest element is reached. In addition, every time the input size doubles, execution time has grown by about nlogn for worst case, average case and best case. In addition, the K index has no effect on execution time, since sorting is done first and then the K'th smallest element is reached.



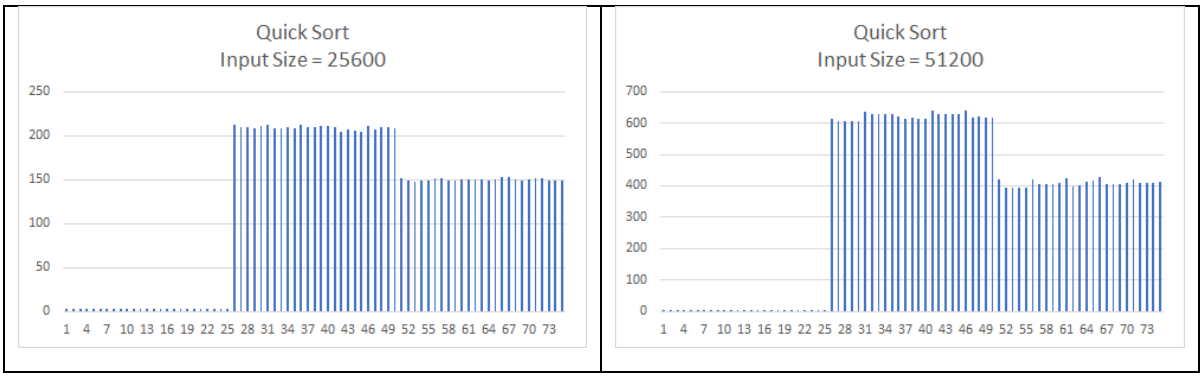


Merge Sort Average Time Complexity = O(nlogn)

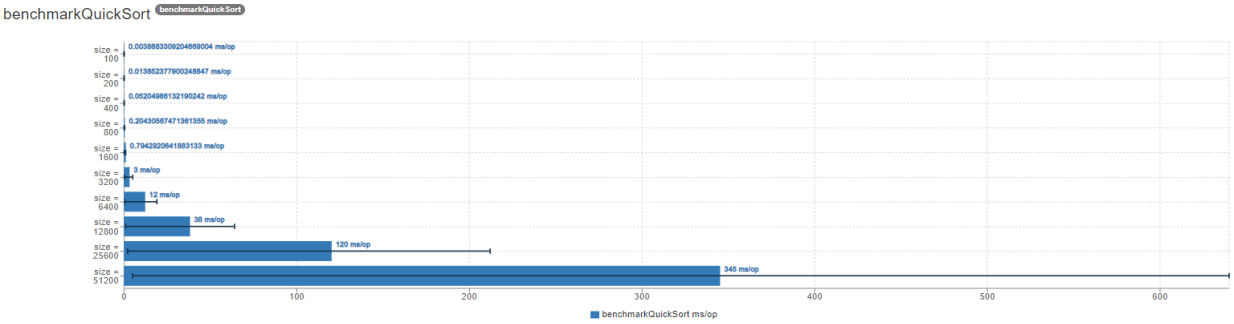


3. Quick Sort

In the quick sort algorithm, the best case for the random input is observed and the random input has the shortest execution time. For ordered and reversed input, execution time was observed as worst case. Considering the graphics, it can be seen that the growth rate is n for the best case and n^2 for the worst cases.

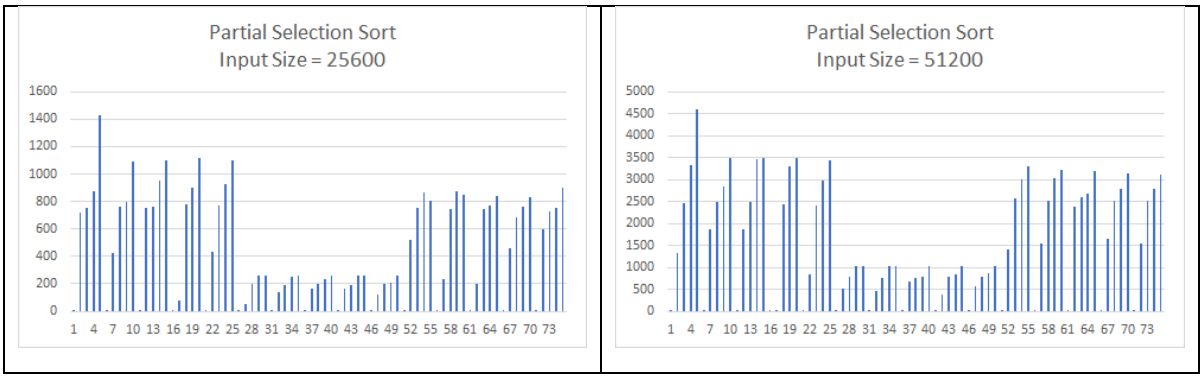


Quick Sort Average Time Complexity = O(nlogn)

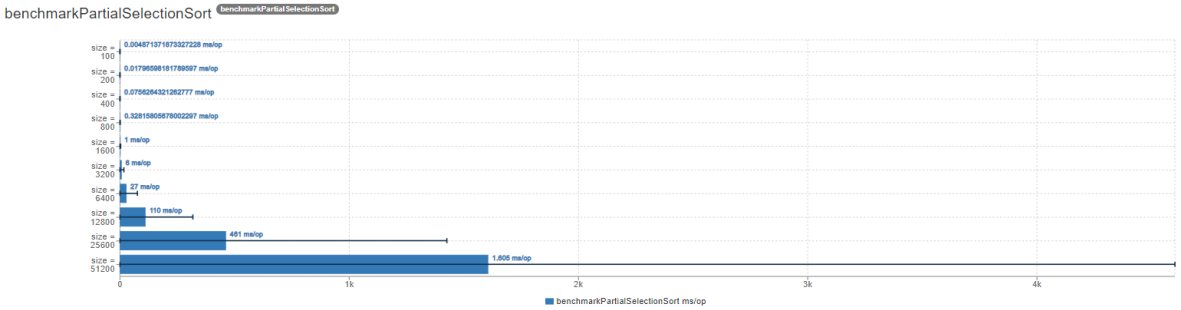


4. Partial Selection Sort

Since it is a partial algorithm, it depends on the K element. The worst case of the algorithm tested in this experiment is when element K is the last element of the list. Data in sequential order shows the best case, while data in reverse order shows the worst case. Random data is around the mean. But this may vary according to the distribution of randomly incoming elements. Best case is 1.

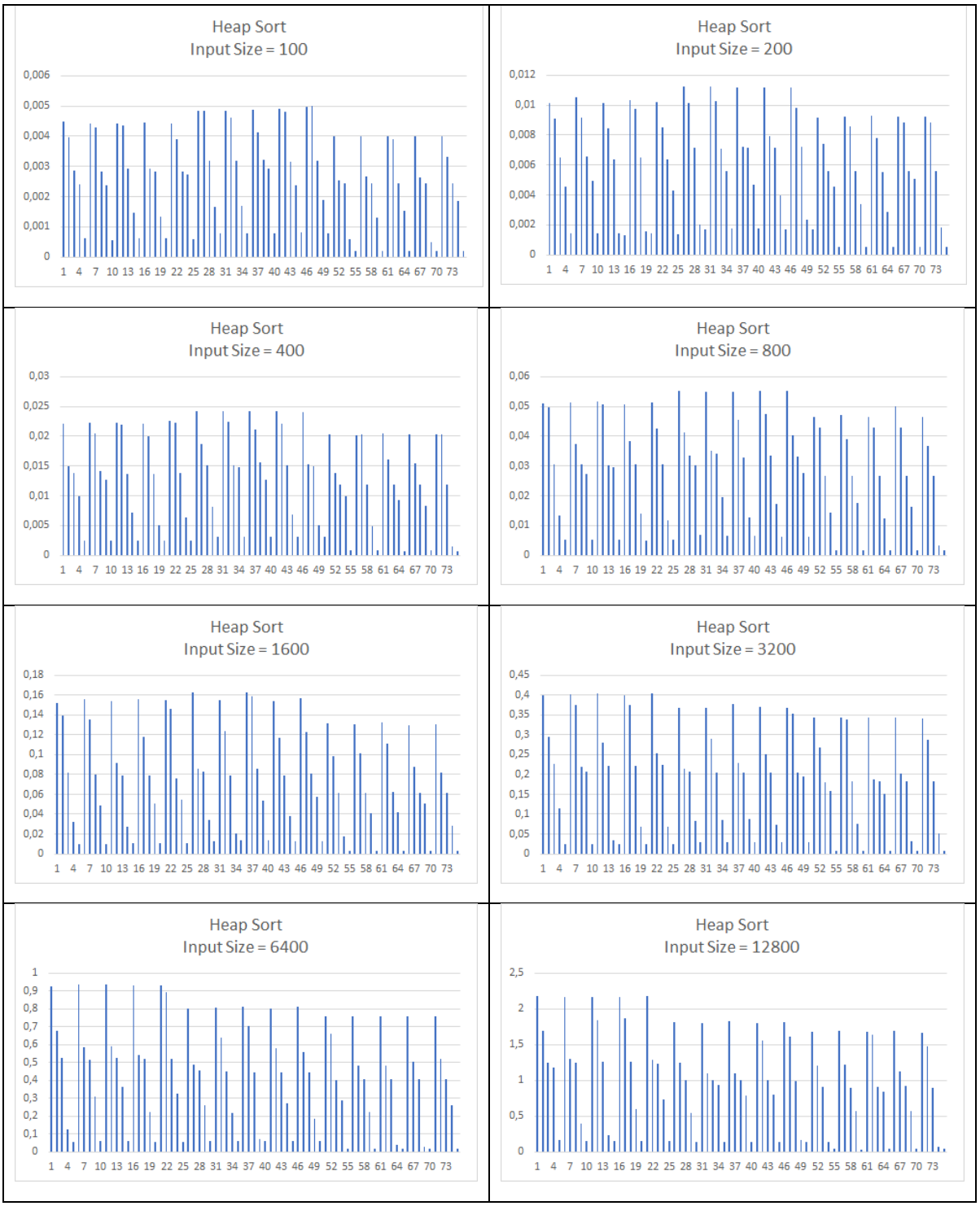


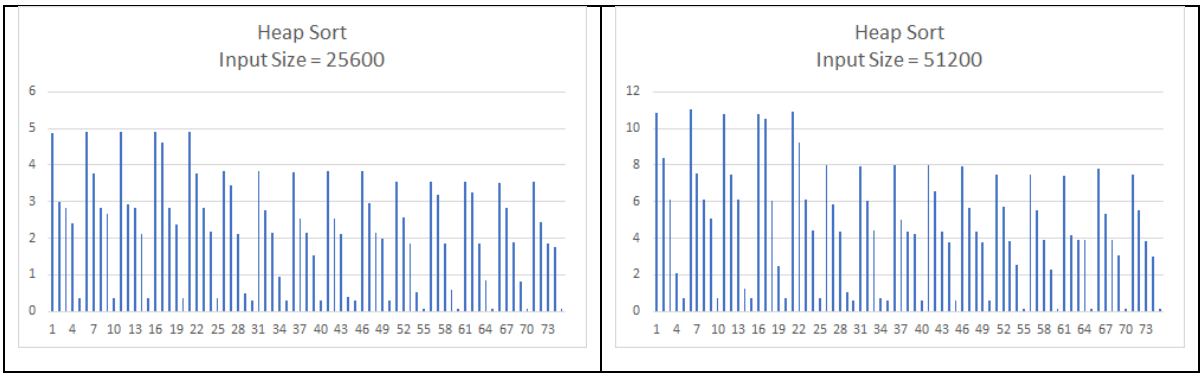
Partial Selection Average Time Complexity = O(kn)



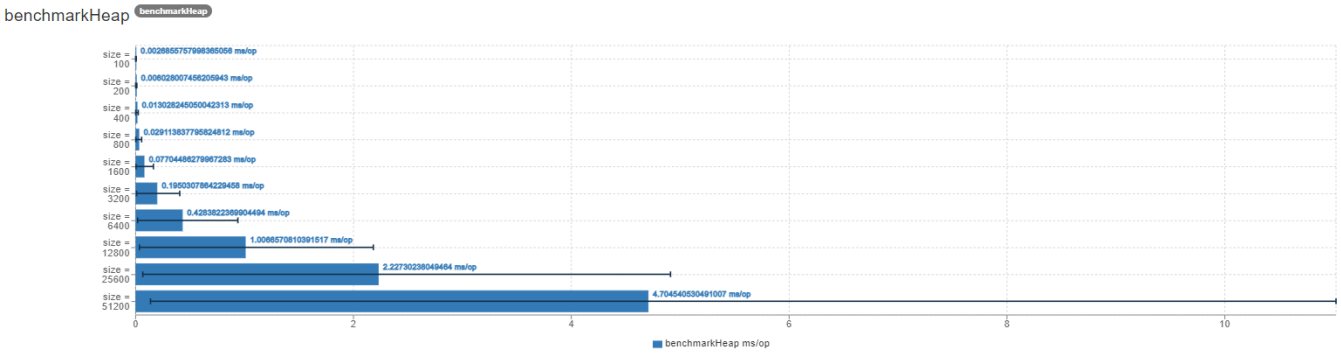
5. Partial Heap Sort

As can be seen in this experiment, the worst case of the algorithm is when element K is the first element of the list. Because of the max heap, the first element will be last replaced. Randomly given values complete the process in a longer time than reverse-ordered or ordered situations. The ordered list, on the other hand, works relatively slower than the reverse-ordered list.



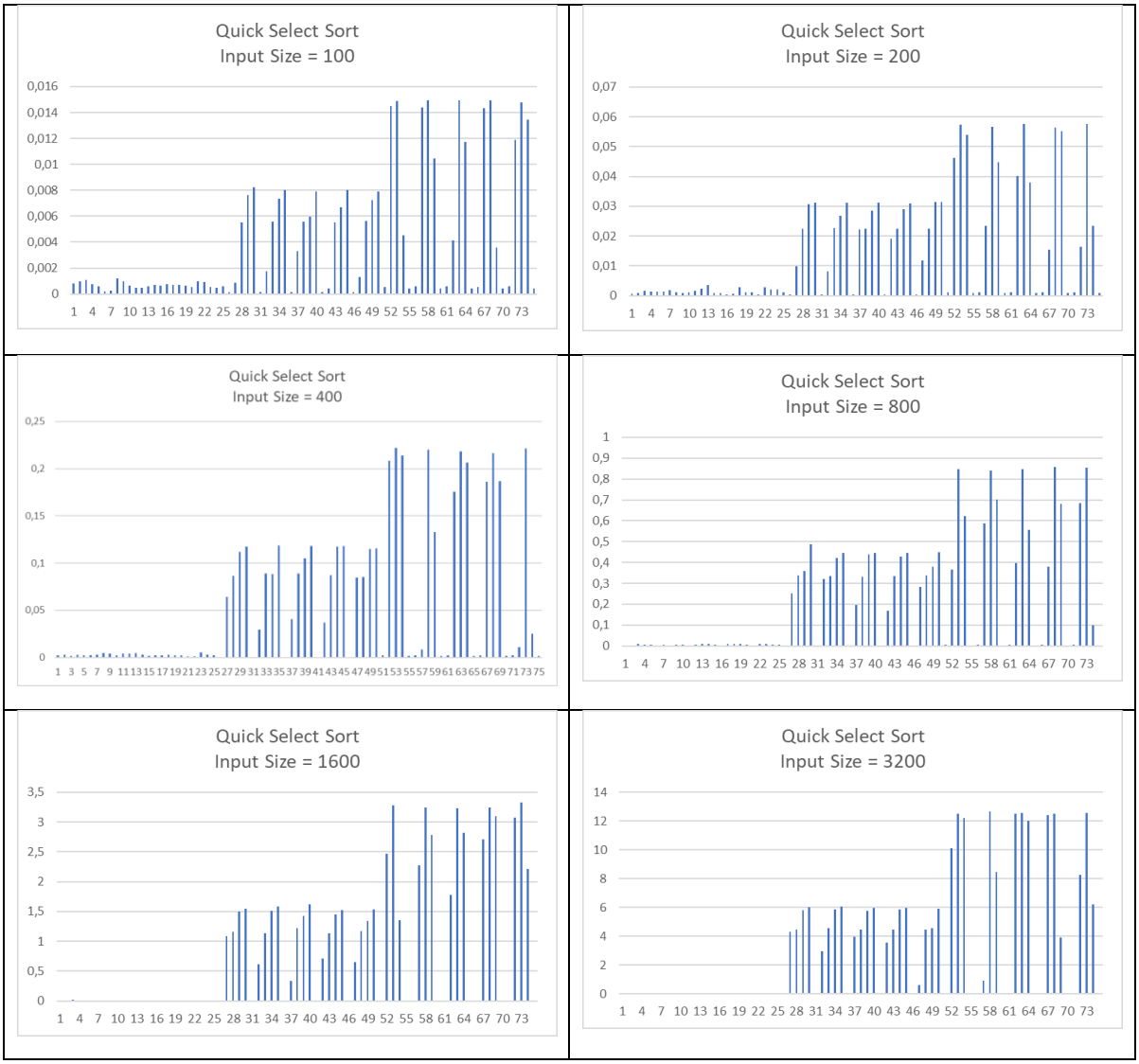


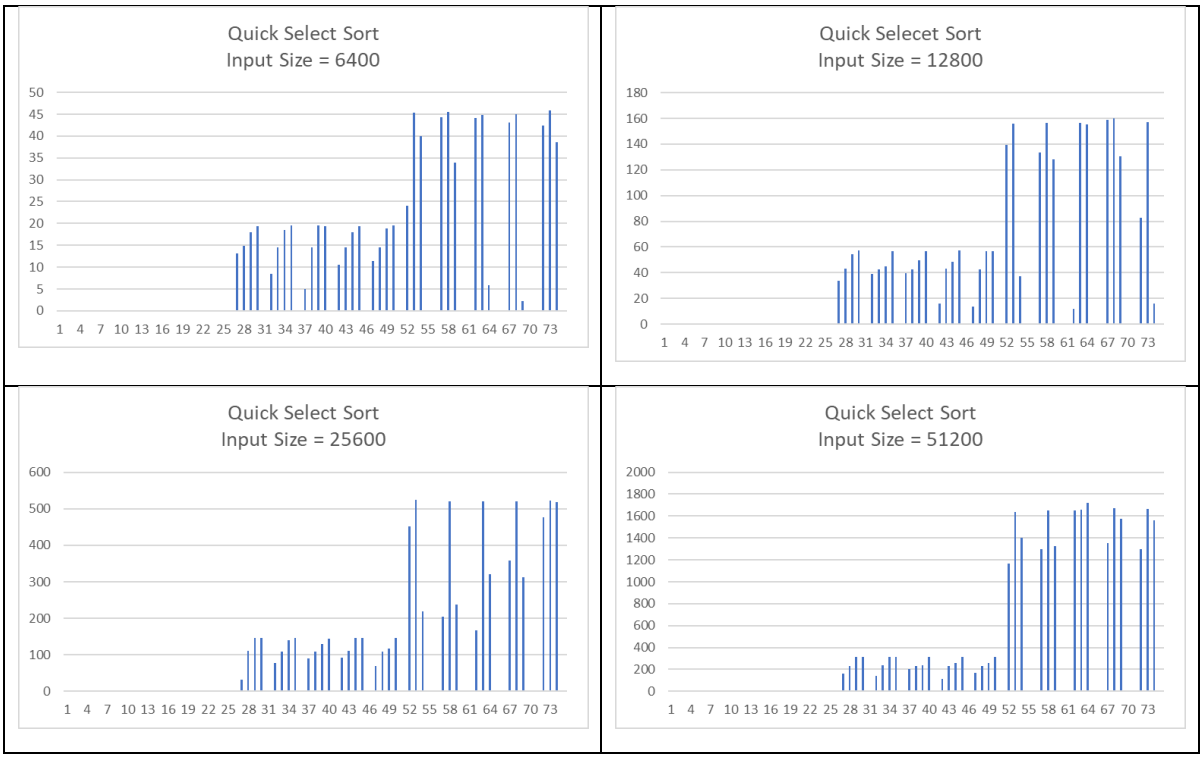
Heap Sort Average Time Complexity = O(nlogn)



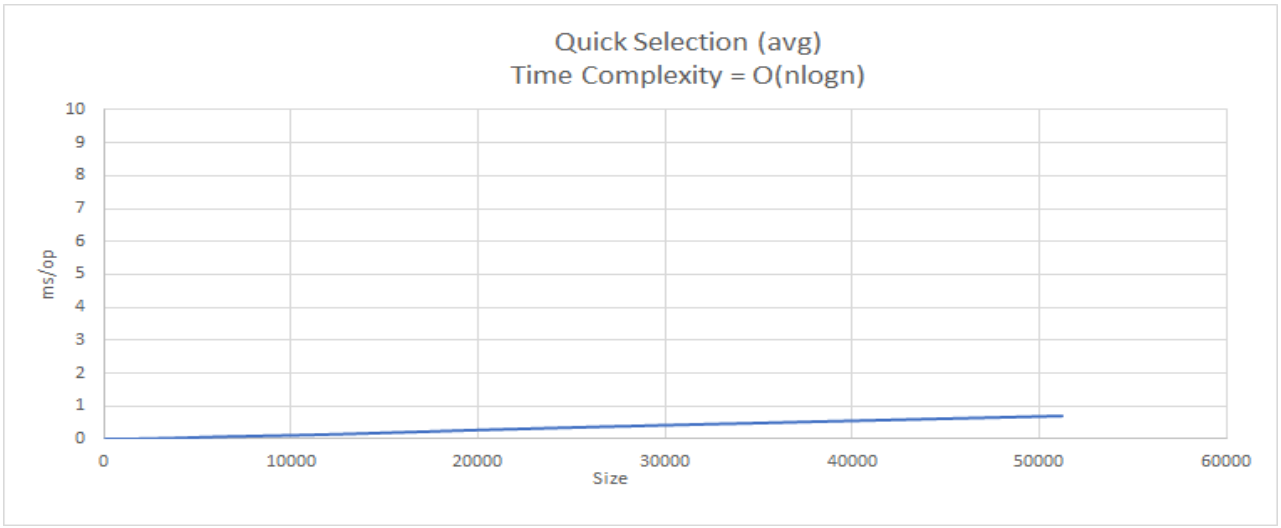
6. Quick Select

In the quick select algorithm, the best case is observed for the random input and the random input has the shortest execution time. For ordered input, execution time was observed as average case and reversed input was observed as worst case. Execution is slower when the index K is in the middle of the input. With the index K at the beginning and at the end, execution is faster than in the middle parts.

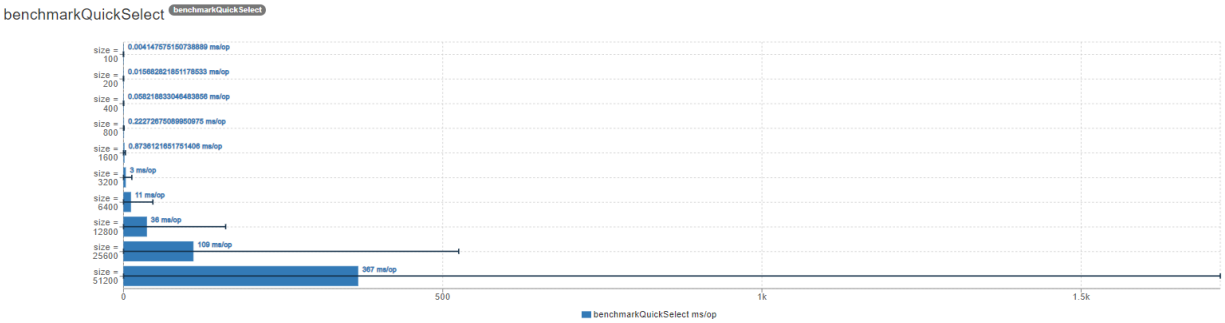




Average case analysis for all input sizes and types for Quick Select Algorithm:



Quick Select Average



7. Median Of Three

This algorithm could not be done.

Total Average For Every Iteration

```
Result "com.App.group":
  518.329 ±(99.9%) 192.852 ms/op [Average]
  (min, avg, max) = (169.461, 518.329, 951.965), stdev = 257.451
  CI (99.9%): [325.477, 711.180] (assumes normal distribution)

Secondary result "com.App.group:benchmarkHeap":
  6.210 ±(99.9%) 2.988 ms/op [Average]
  (min, avg, max) = (0.757, 6.210, 11.123), stdev = 3.989
  CI (99.9%): [3.222, 9.199] (assumes normal distribution)

Secondary result "com.App.group:benchmarkInsertion":
  1099.526 ±(99.9%) 41.788 ms/op [Average]
  (min, avg, max) = (994.970, 1099.526, 1273.688), stdev = 55.786
  CI (99.9%): [1057.738, 1141.314] (assumes normal distribution)

Secondary result "com.App.group:benchmarkMerge":
  5.320 ±(99.9%) 0.095 ms/op [Average]
  (min, avg, max) = (5.189, 5.320, 5.576), stdev = 0.127
  CI (99.9%): [5.225, 5.415] (assumes normal distribution)

Secondary result "com.App.group:benchmarkPartialSelectionSort":
  1993.361 ±(99.9%) 1148.739 ms/op [Average]
  (min, avg, max) = (0.047, 1993.361, 4584.455), stdev = 1533.534
  CI (99.9%): [844.622, 3142.100] (assumes normal distribution)

Secondary result "com.App.group:benchmarkQuickSelect":
  0.755 ±(99.9%) 0.221 ms/op [Average]
  (min, avg, max) = (0.135, 0.755, 1.209), stdev = 0.295
  CI (99.9%): [0.534, 0.976] (assumes normal distribution)

Secondary result "com.App.group:benchmarkQuickSort":
  4.801 ±(99.9%) 0.071 ms/op [Average]
  (min, avg, max) = (4.675, 4.801, 5.067), stdev = 0.094
  CI (99.9%): [4.730, 4.871] (assumes normal distribution)
```

References:

www.geekforgeeks.com

www.stackoverflow.com

www.jmh.morethan.io

www.github.com